Deconstructing RDMA-enabled Distributed Transaction Processing: Hybrid is Better!

Xingda Wei, Zhiyuan Dong, Rong Chen, Haibo Chen

Institute of Parallel and Distributed Systems (IPADS)
Shanghai Jiao Tong University
Remote Direct Memory Access (RDMA)

Kernel bypassing network

- Ultra low latency~(5us)
- Ultra high throughput

Offloading technology (one-sided)

- Bypassing CPU
- Read/Write, CAS\(^2\) server’s memory

Gain interests from Academia & Industry

- Orders of magnitude improvements on distributed applications
- Available in the public cloud\(^1\)

[2] Atomic compare and swap
On-going debate over how to use RDMA for TXs

- **Get(A)**
  - Coordinator
  - A’s store

One-sided READ(I)
- Lookup A
- Read A

Two-sided RPC(II)
- RPC request
- RPC reply

Performance
- ✓
- ✗

#Round-trips
- >= 2
- 1
Transaction(TX)s are more complex

TX (e.g. OCC\textsuperscript{[1]}) uses multiple phases for serializability & availability

- Each can be offloaded with one-sided primitive

\[
\text{TX} \{ A = A + 1 \}
\]

\begin{tabular}{l|c|c|c|c}
 & \textit{Execution} & \textit{Validation} & \textit{Logging} & \textit{Commit} \\
\hline
\textbf{Coordinator} & \begin{itemize}
\item One-sided READs
\item Write log
\end{itemize} & \begin{itemize}
\item CASs + READs
\item WRITEs
\end{itemize} & \begin{itemize}
\item WRITEs
\end{itemize} & \begin{itemize}
\item WRITEs
\end{itemize} \\
\hline
\textbf{A’s primary} & \begin{itemize}
\item Execute TX’s logic
\end{itemize} & \begin{itemize}
\item Verify execution’s consistency
\end{itemize} & \begin{itemize}
\end{itemize} & \begin{itemize}
\end{itemize} \\
\hline
\textbf{A’s backup} & \begin{itemize}
\end{itemize} & \begin{itemize}
\end{itemize} & \begin{itemize}
\end{itemize} & \begin{itemize}
\end{itemize} \\
\hline
\end{tabular}

\textsuperscript{[1]} Optimistic concurrency control
Transaction(TX)s are more complex

Protocols
- OCC, 2PL, SI,....

Impl on hardware devices
- One-sided vs. Two-sided, ...
- CX3, CX4, CX5, ROCE

OLTP workloads
- TPC-C, TPC-E, TATP, Smallbank, ...
This work: how to use RDMA for TXs

Focus on **OCC** in this work
- Use **phase-by-phase** approach

**Well-tuned** RDMA execution framework
- Representative RNICs (CX3 - CX5)

Representative OLTP workloads
- TPC-C, TPC-E, and Smallbank

---

**Implements & Hardware**

- **Optimistic Concurrency Control**
  - Widely used in Centralized
    - Silo[SOSP’13]  Foedus[SIGMOD’15]
    - ...  
  - Distributed
    - FaRM[SOSP’15]  TAPIR[SOSP’15]
    - ...
Phase-by-phase analysis is effective & useful

OCC uses **consecutive** phases

- Better **phase** performance -> Better **overall** performance

**TX** \( \{ A = A + 1 \} \)

- **Coordinator**
- **A's primary**
- **A's backup**

**Execution**
- Read
- \( V \)

**Validation**
- Lock
- Read

**Logging**
- Write

**Commit**
- Write + Unlock

Like **query** in graph; **get()** in key-value store

Like **write** a file in distributed file system, etc
Deconstructing TX with phase-by-phase analysis

OCC uses **consecutive** phases

- Better **phase** performance -> Better **overall** performance

---

DrTM+H

*No* single primitive wins *all* the time!

---

https://github.com/SJTU-IPADS/drtmh
Outline

- RDMA primitive-level analysis
- Phase-by-phase analysis for TX
- DrTM+H: Putting it all together
System model & evaluation setup

Evaluation setup

<table>
<thead>
<tr>
<th>16 x</th>
<th>CPU</th>
<th>RNIC</th>
<th>Link layer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>24 cores</td>
<td>2 * ConnectX-4 RNIC</td>
<td>Infiniband</td>
</tr>
</tbody>
</table>
Primitive analysis

One-sided primitive

- Simple implementation (Native verbs API)
- Optimized event loop (Async communication)

Two-sided (RPC)

- FaSST RPC [OSDI’16]
- Fastest in our setting

Throughput (million reqs/second)

- READ: 90
- WRITE: 103
- CAS: 48
- Two-sided: 80

**Client**

**Server**

**Note:**

- READ/WRITE is faster with known address
- CAS is slower, but with sufficient performance (48M per machine)
Passive ACK (PA)

**Opt:** when the reply is not on the critical path of the execution

One-sided primitive
- **Unsignaled** requests

Two-sided primitive
- **Batch** replies (passively)

Client
- Req
- Reply

Server
- +PA
- Req
- Reply

Throughput (millions reqs/second)

- **Two-sided is faster w PA**
- Better

PA **usually not** apply to READ/CAS

- 90
- 103
- 48
- 80
- 133
- 160

- READ
- WRITE
- CAS
- Two-sided
Towards phase-by-phase analysis

Transactional system

- Built atop of our well-tuned execution framework (primitive analysis)

Workloads

- TPC-C/no: new-order (distributed)
- Smallbank
- TPC-E/cp: custom-position
Execution = READs

TX{A = A + 1} = Coordinator
A's store

One-sided (I) Cache

Optimization for one-sided primitive
- RDMA friendly store (e.g. DrTM-KV) \(\rightarrow\) \(~\) One-round lookup
- Index cache, cache hot items address \(\rightarrow\) One-round (lookup + read)

Two-sided (II)

RPC request
RPC reply

Exe | Val | Log | Commit
Execution = READs

- **Better**
  - Two-sided
  - One-sided
  - One-sided/Cache

- **Latency (ms)**
  - Underloaded
  - Overloaded

- **Throughput (millon TXs/second)**

**Hybrid usage of ONE/Cache + Two-sided(miss)**

- Two-sided is faster with one round-trip
- One-sided is faster with high CPU utilization
- READs are better with one round-trip.
Validation = LOCKs + READs

Optimization for one-sided primitive (for one round-trip)

- Address known w the execution phase -> no need for lookup
- Locked value cannot be changed -> doorbell batch READs w CASs
Validation = LOCKs + READs

**Graph:**
- **Axes:**
  - Y-axis: Latency (ms)
  - X-axis: Throughput (million TXs/second)
- **Data Points:**
  - Two-sided: Blue squares
  - One-sided: Red circles

**Legend:**
- Green square for CAS + Read
- Orange square for batched together

**Annotation:**
- Validation is **suitable** for one-sided because of **one round-trip**
- Validation = LOCKs + READs

**Process:**
- Exe
- Val
- Log
- Commit

**Process Steps:**
- Execution
- Validation
- Logging
- Committing

**Validation:**
- Validation is performed with LOCKs and READs together.

**CAS + Read Batched Together:**
- CAS and read operations are batched together for efficiency.

**Notes:**
- Better values indicate lower latency and higher throughput.

---

**Slide Number:** 17
Logging = WRITEs

One round-trip for one-sided primitive

- **Ring buffer** based log management [FaRM@NSDI’14]
- **RNIC ack** -> logging succeed (Totally **bypassing** CPU)
Logging = WRITEs

Throughput (million TXs/second) \(\rightarrow\) Better

- Two-sided
- One-sided

Latency (ms)

Better

- LOGGING can be always offloaded with one round-trip
- LOGGING is better with one-sided

Exe | Val | Log

Commit

rep-factor=2

One-sided (I)
Commit = WRITEs + UNLOCKs

One-sided (I)

TX{A = A + 1}

Coordinator

A’s store

Unlocks implemented as WRITEs

One round-trip for one-sided primitive

- Address known w the execution phase -> no need for lookup

Adding passive ACK to both primitives

- Log succeed indicates TX’s commit

Two-sided (II)

RPC request

RPC reply

Log succeed indicates TX’s commit
Commit = WRITEs + UNLOCKs

- Two-sided w PA is faster
- Two-sided w PA has higher peak throughput
- Commit RPC costs is small
- Two-sided saves CPU at sender

Better

Latency (ms)

Throughput (million TXs/second) ⇒ Better

Commit = WRITEs + UNLOCKs

Exe | Val Log Commit
DrTM+H: Hybrid is better!

Hybrid system supports **serializability** & **high availability**

Specific optimizations

- **Passive ACK** to the commit phase ( & log cleaning message)
- Speculative execution to send **outstanding requests (OR)** from one TX
Performance & scalability on TPC-C/no

DrTM+H scales well @ an (emulated) 80-node connections
End-to-end comparison against prior designs

In the **same platform**, the **same protocol**, but w **different choices**

<table>
<thead>
<tr>
<th></th>
<th>E</th>
<th>V</th>
<th>L</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>FaSST-OCC</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>II</td>
</tr>
<tr>
<td>DrTM+R</td>
<td></td>
<td></td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>FaRM</td>
<td></td>
<td></td>
<td>+</td>
<td>II</td>
</tr>
<tr>
<td>DrTM+H</td>
<td></td>
<td>+</td>
<td>II</td>
<td>I</td>
</tr>
</tbody>
</table>


---

[1] FaSST uses a simplified OCC protocol compared to FaRM & DrTM+R.

---

**Throughput (million TXs/second)**

**Better**

**Latency (ms)**

- **FaSST-OCC**
- **FaRM**
- **DrTM+R**
- **DrTM+H**

- **Index cache**
- **Logging**
- **Validation**
Where do the performance gains come from?

- **90th Latency (ms)**
  - Base (Two)
  - + One READ
  - + One Log
  - + Index Cache
  - + One VAL
  - + PA
  - + OR

- **Throughput (KTX/sec)**
- **Peak throughput**

- **1 concurrent req / server**
- **16 concurrent req / server**
Not a hard conclusion!

May depends on RNIC’s characteristic & network setting

The results start from the \textit{primitive level analysis}.

[1] 1-way replication used due to cluster limitation

[2] Main results in this talk

[3] 1-RNIC per machine, others uses 2
Evaluation summary

**Offloading** w one-sided *improves the performance*

- Especially *w/o adding more round-trips*
- Less affected by **CPU load** at the server

**One-sided primitive has good scalability on modern RNIC**

- Especially when RNIC is **not the bottleneck** of the application
- Although one-sided primitive is restricted by hardware limitation
More: check our paper!

- Optimized execution framework
- Results of large scale
  Modern RNIC has good scalability for one-sided primitive
- Read-only Transactions
  A hybrid scheme also wins
- TPC-E, Smallbank
Conclusions

The first systematic study on

- How to use RDMA for OCC TXs

No single primitive is better!

- Depends on workload pattern & primitive analysis

Execution framework & DrTM+H are available @

https://github.com/SJTU-IPADS/drtmh
Backups
Improved overall systems

FaSST’s simplified OCC protocol

Adding hybrid-schema for logging
**Smallbank workloads**

<table>
<thead>
<tr>
<th></th>
<th>FaSST-OCC</th>
<th>FaRM</th>
<th>DrTM+R</th>
<th>DrTM+H</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CX3[1]</strong></td>
<td>![Symbol]</td>
<td>![Symbol]</td>
<td>![Symbol]</td>
<td>![Symbol]</td>
</tr>
<tr>
<td><strong>CX4</strong></td>
<td>![Symbol]</td>
<td>![Symbol]</td>
<td>![Symbol]</td>
<td>![Symbol]</td>
</tr>
<tr>
<td><strong>CX4-ROCE[1]</strong></td>
<td>![Symbol]</td>
<td>![Symbol]</td>
<td>![Symbol]</td>
<td>![Symbol]</td>
</tr>
<tr>
<td><strong>CX5[1][2]</strong></td>
<td>![Symbol]</td>
<td>![Symbol]</td>
<td>![Symbol]</td>
<td>![Symbol]</td>
</tr>
</tbody>
</table>

[1]1-way replication used due to cluster limitation
[2]1-RNIC per machine, others uses 2
RDMA based execution framework

Applied & based RDMA optimizations

- FaRM [NSDI’14, SOSP’15]
- Herd [NSDI’14]
- RDMA guideline [ATC’16]
- FaSST [OSDI’16]

Others

- LITE [SOSP’2017] -> Further improve one-sided's scalability
Results using large connections

16 node

Emulate 80-node
Comparison of two-sided implementations

FaSST RPC uses UD SEND/RECV
RDMA enabled application

Load balance framework

Distributed TXs

Graph processing systems

Distributed file system